Banco di prova:
10 Piastre di registrazione

N. 60 Giugno '90
L. 7000 - Frs. 10,5

fare ELETTRONICA

Realizzazioni pratiche • TV Service • Radiantistica • Computer hardware

REALIZZAZIONI PRATICHE

Segreteria telefonica digitale

Intercom per motociclisti

Pseudo stereo per TV

COMPUTER HARDWARE

Cartuccia C64 senza EPROM

Analizzatore logico seriale

RADIANTISTICA

Ricevitore c.d. 14 MHz

IRRIGATORE ELETTRONICO

IN COLLABORAZIONE CON ETI ELECTRONICS TODAY INTERNATIONAL

TV SERVICE

EUROPHON CTV3699-5199

Taxe Percue (Tassa Riscossa) Milano Ferr. Corr.
Spedizione in Abb. Post. Gruppo III/70

Una cartuccia C64 senza EPROM

Con questa speciale tecnica, non avrete bisogno di nessun costoso dispositivo di programmazione per preparare le vostre cartucce per C64 o C128.

Le cartucce sono utili e facili da usare. I programmi su cartuccia si caricano istantaneamente. Potrete costruire una cartuccia utilizzando EPROM (memorie di sola lettura cancellabili e programmabili) per circa 30.000 lire, se siete oculati nell'acquisto. Ma le EPROM devono essere programmate o "bruciate" usando un apposito programmatore, che costa circa 175.000 lire. Se fate qualche errore o volete modificare i programmi, avrete bisogno di un cancellatore di EPROM, altre 60.000 lire. L'economica cartuccia EPROM costa quindi poco meno di 300.000 lire di spese d'impianto.

Un'alternativa consiste nell'utilizzare la RAM (memoria ad accesso casuale) in luogo delle EPROM. La RAM può essere programmata dallo stesso computer e le informazioni contenute possono essere modificate in ogni istante, senza speciali apparecchiature supplementari.

Il problema con la RAM è che tutto quanto vi è stato memorizzato scompare quando si interrompe l'alimentazione: non è esattamente ciò che ci si attende da una cartuccia. Utilizzando però le speciali RAM CMOS (semiconduttori complementari ad ossido metallico), che assorbono una limitatissima corrente di riposo, potremo usare una piccola batteria per conservare intatte le informazioni contenute nella RAM. La memoria viene conservata anche quando il computer è spento e la cartuccia estratta. La 4464-15s, costruita dalla NEC Corp. ed utilizzata in questo progetto, assorbe normalmente una corrente di riposo di 0,1 µA: una batteria con le dimensioni di una moneta da 500 lire può alimentarla per parecchi anni.

Figura 1.
Tutti i riferimenti tra parentesi sono numeri di piedini per la porta di espansione del C64: vedere sulla guida di riferimento per i programmatori del C64.

Costruzione della cartuccia RAM

Per il nostro prototipo, abbiamo usato una scheda "preforata" Vector 3791-5, sulla quale ci sono 44 piste già incise (22 su ciascuna faccia), con la giusta spaziatura per allinearsi con la presa di espansione del C64. Se possedete l'attrezzatura per incidere i circuiti stampati, questa potrebbe essere la scelta più economica. Potreste anche adattare la scheda di una vecchia cartuccia, oppure acquistarne una prevista per essere usata nel C64. Accertatevi che sulla scheda utilizzata siano disponibili le linee di indirizzamento A13-A15 (piedini F, H e J): non sono necessarie per le cartucce EPROM e possono essere assenti sulle schede progettate per questo scopo.

Per montare il nostro circuito, abbiamo usato il "wire wrap", ma qualunque sistema di cablaggio andrà bene. Per i circuiti integrati sono consigliati, ma non indispensabili, gli zoccoli. Attenersi alle consuete precauzioni quando si lavora con le RAM CMOS, perché un maneggio improprio potrebbe danneggiarle irreparabilmente.

La Figura 1 mostra lo schema di una car-

tuccia RAM da 8 K. La Figura 2 mostra i circuiti necessari per aggiungere altri 8 K. L'interruttore S1 controlla l'alimentazione delle RAM CMOS. Se l'interruttore è chiuso, arriva l'alimentazione dal C64.
Con S1 aperto od il computer spento, entra in funzione la batteria, che conserva i dati nella memoria. S2 controlla la lettura/scrittura dei dati nelle RAM. Con questo interruttore chiuso, il computer può modificare i dati. L'apertura di S2 fa apparire la RAM al C64 come se fosse una ROM.
S3 ed S4 permettono alla cartuccia RAM di emulare i tre tipi di cartuccia usati con il C64, come vedremo tra poco. S5 è utilizzato soltanto con la versione da 16 K: permette di "spostare" gli 8 K superiori della RAM verso una zona dove possano essere programmati. I diodi eliminano elettricamente la batteria dal circuito quando il computer fa arrivare l'alimentazione, evitando che le pile tentino da sole di alimentare l'intero C64. I diversi resistori stabiliscono i valori di base per le linee di segnale e commutano le RAM nella condizione di corrente di riposo quando viene aperto S1.
Il 74LS42 è un decodificatore che controlla lo stato delle tre linee con indirizzo più elevato (A13-A15) e produce una diversa uscita per ciascuna combinazione di questi indirizzi.
Ci sono 8 uscite e pertanto possiamo scegliere con questo chip 8 banchi di memoria da 8 K.
I condensatori C1 e C2 vengono utilizzati per eliminare qualsiasi disturbo dalla linea di alimentazione. C1 deve essere montato vicino al bordo della scheda che si inserisce nel computer e C2 più vicino possibile al 74LS42.
Potrete trovare altre RAM da 8 x 8 K con analoghe caratteristiche di assorbimento a riposo.
Se hanno un tempo di accesso di 150 ns o meno, vanno bene per questa applicazione. Richiedere sempre i relativi fogli dati: le piedinature potrebbero essere diverse da quelle indicate nei nostri schemi.
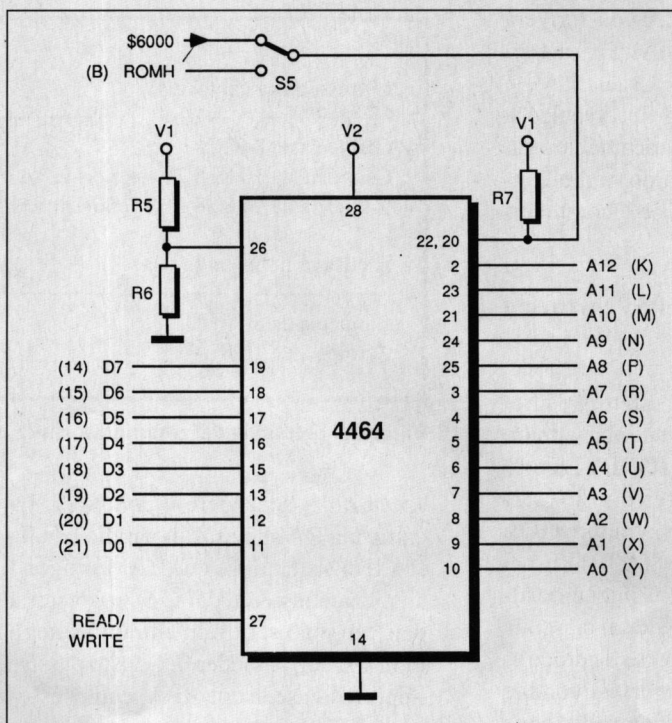
Figura 2. Parti supplementari necessarie per una cartuccia da 16 K.

Come funziona la cartuccia

Il C64 utilizza un PLA (array logico programmato) per controllare l'accesso di RAM, ROM e cartucce ai bus degli indirizzi e dei dati. Le cartucce possono avere tre configurazioni ed il PLA identifica il tipo di cartuccia mediante due linee di controllo, chiamate "GAME" (piedino 8) ed "XROM" (piedino 9). La cartuccia RAM utilizza gli interruttori S3 ed S4 per attivare le linee di controllo. Una cartuccia da 8 K appare sempre nel campo di indirizzamento che va da $8000 a $9FFF. Dispone anche di un ponticello interno, che manda a livello basso la linea XROM. La chiusura di S4 simula questa configurazione. Una cartuccia da 16 K ha ancora 8 K tra $8000 e $9FFF. Gli 8 K superiori possono risiedere in una tra due altre aree. Se è bassa solo la linea GAME (S3 chiuso, S4 aperto), gli 8 K superiori vengono disposti tra $E000 e $FFFF. Se sono a livello basso sia GAME che XROM (S3 ed S4 chiusi), tutti i 16 K saranno contigui, tra $8000 e $BFFF. Una cartuccia da 8 K contiene normalmente un programma autonomo, oppure uno che utilizza il BASIC e le routine ROM Kernal incorporati nel C64. Una cartuccia da 16 K nel campo di indirizzamento tra $8000 e $BFFF sostituisce la ROM BASIC. Gli 8 K superiori possono contenere un BASIC modificato e quelli inferiori le estensioni BASIC. La terza configurazione è stata concepita esclusivamente per i giochi. In questo modo, il chip VIC terrà conto del set di caratteri nella parte superiore della memoria, da $E000 a $FFFF. Questo semplifica la grafica a bassa risoluzione per i giochi, ma è inadatto come sostituto del Kernal. I programmi di queste cartucce devono perciò essere del tutto autonomi.
Tutti i chip di memoria, RAM o ROM, sono commutati sui bus dei dati e degli indirizzi mediante le linee "chip select". Nel C64, è il PLA a controllare queste linee e pertanto decide la scelta tra la RAM, una delle ROM del sistema, oppure la cartuccia. Se il PLA rileva che è stata inserita una cartuccia (tramite le linee GAME ed XROM) ed il microprocessore emette un comando "READ", allora viene scelta la zona di memoria della cartuccia. Il PLA controlla questa selezione, tramite le linee "ROML" (piedino 11) e "ROMH" (piedino B). Se viene emesso un comando "WRITE", il PLA esclude la memoria della cartuccia e porta invece a quegli indirizzi la RAM. Nessuna cartuccia Commodore contie-

ne RAM, pertanto il PLA non scriverà nella nostra cartuccia RAM. Per ottenere questo risultato, si bypassa il PLA e si effettua la propria decodifica. Qualcosa viene eseguito automaticamente dal chip 74LS42 e qualcosa dovremo controllarlo manualmente, con il commutatore S5.

Programmazione della cartuccia RAM

Quando il C64 è acceso, effettuare il reset con l'interruttore esterno; oppure premendo il tasto "RESTORE". Le routine nella ROM Kernal cercano una cartuccia. Tutte le cartucce avranno 8 K, a partire dalla locazione $ 8000. Il Kernal cerca il codice "CBM80", iniziando all'indirizzo $ 8004. Il bit alto di ciascuna lettera deve esser settato. Se il codice esiste, verranno saltate le normali routine di inizializzazione ed il controllo viene trasferito al programma della cartuccia. Al reset all'accensione o via hardware, l'indirizzo memorizzato nell'ordine basso-alto in $ 8000/$8001 viene utilizzato per un salto indiretto. Se è stato premuto "RESTORE", viene usato invece l'indirizzo memorizzato in $8002/$8003.
Per creare nella cartuccia un programma di auto-avviamento, dovrete installare la frase di codice ed i giusti indirizzi.
Potrete anche trovarvi nella necessità di richiamare qualcuna delle routine di inizializzazione saltate. Potrete memorizzare codici macchina nella cartuccia RAM senza frase di auto-start e SYS al codice, dal BASIC od in modo diretto, invece di effettuare l'auto-start. Se volete utilizzare la cartuccia RAM per memorizzare uno dei vostri programmi BASIC, ricorrete al listato 1. Un RUN di questo programma crea un file chiamato "RAMCART" sul dispositivo 8, destinato al disco. Potrete modificare questi default nella riga 100 del programma. Il codice sorgente del file è mostrato, in formato PAL, nel listato 2.
Per usare il programma installare la cartuccia RAM, chiudendo poi S1 ed S2. Accertarsi che S3 ed S4 siano entrambi

	S1	S2	S3	S4	S5
Lettura dalla cartuccia:					
Cartuccia da 8 K	ON	OFF	OFF	ON	X
Cartuccia da 16 K, 8 K superiori in $A3000	ON	OFF	ON	ON	ROMH
Cartuccia da 16 K, 8 K superiori in $AE000	ON	OFF	ON	OFF	ROMH
Scrittura nella cartuccia:					
Cartuccia da 8 K	ON	ON	OFF	OFF	X
Cartuccia da 16 K	ON	ON	OFF	OFF	$6000

Figura 3. Settaggio dei commutatori per usare la cartuccia.

aperti e poi accendere il computer. La cartuccia RAM è ora "in parallelo" alla RAM di sistema. Le due RAM vengono analizzate insieme dal C 64 e gli stessi dati vengono scritti, in entrambe, negli indirizzi corrispondenti. Questo passo è importante: se le due RAM contenessero dati diversi, entrerebbero in conflitto sul bus dei dati. Caricare (con LOAD) il programma "RAMCART" con ",8,1". In questo modo il codice viene piazzato all'inizio della memoria della cartuccia RAM. Caricate ora (LOAD) il programma BASIC che volete memorizzare ma non date RUN.
Battete invece: SYS 32882
Il codice macchina memorizzato da "RAMCART" copierà il programma BASIC nella RAM della cartuccia. Se il programma fosse troppo lungo, maggiore di 31 blocchi del disco, apparirà un messaggio di errore. Quando appare la richiesta "READY", aprire S2. In questo modo si stacca la cartuccia dalla linea READ/WRITE ed allora i dati non potranno più essere modificati dal computer. Spegnere il C 64: la batteria conserverà il programma nella RAM della cartuccia. Chiudere S4 per informare il PLA che si tratta di una cartuccia da 8 K e riaccendere il computer. Il codice di auto-start nella cartuccia RAM farà inizializzare normalmente il BASIC del sistema. Ricopierà poi il vostro programma nell'area della memoria BASIC. Il comando "RUN" verrà inserito nel buffer della tastiera ed il computer lo eseguirà, facendo partire il vostro programma. La combinazione RUN-STOP/RESTORE vi porterà fuori dal vostro programma BASIC, visualizzando il messaggio "READY". Per riavviare il programma nella cartuccia usare un interruttore di reset hardware oppure impostare: SYS 64738 Per programmare gli 8 K superiori di RAM in una cartuccia da 16 K, è necessaria una tecnica diversa. Dovremo usare la linea ROMH proveniente dal PLA per selezionare la memoria della cartuccia, perché altrimenti il PLA inserirebbe la ROM di sistema. Però il PLA non ci lascerà scrivere dati nella memoria selezionata da ROMH. S5 commuta la linea di selezione degli 8 K di RAM superiori tra l'uscita ROMH del PLA e l'uscita $6000-$7FFF proveniente dal 74LS42. Con S5 in posizione $6000 potrete modificare gli 8 K superiori di dati scrivendo nella RAM in questa locazione inferiore. Riportando S5 in posizione ROMH si farà commutare il PLA della RAM a $A000 oppure $E000 a seconda delle posizioni di S3 ed S4.
Ad esempio, per cambiare il BASIC, inserire una cartuccia RAM da 16 K nel computer, chiudere S1 ed S2, aprire S3 ed S4 e portare S5 nella posizione $6000. Accendere il computer, fare il LOAD di un monitor in linguaggio macchina, che risieda al di sotto di $6000 o al di sopra di $C000 ed usarlo per copiare la ROM BASIC nella RAM in $6000. Usare il modo di verifica della memoria per guardare cosa c'è nei 9 byte che ini-

ziano da $6378. Si tratta del testo "READY", seguito da un "RETURN" ($0D), da un'interlinea ($0A) e da un byte zero di terminazione ($00).Usare il monitor per modificare il testo. Aprire ora S2 per bloccare le modifiche nella RAM e spegnere il computer. Spostare S5 nella posizione ROMH. Chiudere S3 ed S4. Si ordina in questo modo al PLA di disporre gli 8 K di RAM con il BASIC modificato nell'area di indirizzamento normalmente utilizzata dalla ROM BASIC. Accendendo ora il computer, si vedrà il prompt READY modificato e soltanto 30.719 byte liberi, perché anche gli 8 K inferiori della cartuccia RAM vengono inseriti dal PLA. Potrete utilizzare gli 8 K inferiori per inserirvi programmi BASIC oppure estensioni, oltre a qualsiasi modifica da voi apportata al sistema operativo BASIC. I posizionamenti dei commutatori per la programmazione e l'utilizzo della cartuccia sono riassunti in Figura 3. La cartuccia RAM è del tutto compatibile con le schede di espansione, che permettono l'inserimento contemporaneo di diverse cartucce. Accertatevi di aprire S1 quando selezionate una diversa cartuccia, in modo che la RAM in $8000 venga rimossa dai BUS. Potete usare la cartuccia RAM anche su un C128. Nel modo C128, non vengono usate le linee GAME ed XROM: la MMU (unità di gestione della memoria) cerca invece un codice diverso. Dovrete scrivere una routine auto-boot per C128, per poi installarla con la stessa procedura descritta in precedenza per il C64. Questa cartuccia RAM sarà certo per voi un'alternativa poco costosa all'acquisto di un programmatore e cancellatore di EPROM per produrre le vostre cartucce. Anche se possedete già un programmatore di EPROM, la facilità e la rapidità di modifica al software contenuto nella cartuccia RAM rappresenteranno senz'altro un vantaggio.

ELENCO DEI COMPONENTI

B1	**batteria da 3 V (vedi testo)**
C1-2	**cond. ceramici a disco da 50 nF 12 Vl**
D1/4	**diodi 1N4148**
R1-3/5-7	**resistori da 2,2 kΩ**
R2-6	**resistori da 22 kΩ**
S1/4	**gruppo di interruttori DIL**
S5	**deviatore unipolare miniatura**
1	**74LS42 decodificatore BCD 1 da 10**
1	**4464 RAM statica CMOS**

Listato 1

```
1000 rem save "0:ramcart.ldr",8
1010 rem ** by: john bush and noel nyman - seattle, wa
1020 rem ** auto-start support prg
1030 rem ** for c64 ram cartridge
1040 :
1050 rem ** this program will create
1060 rem ** a load ",8,1" module on
1070 rem ** disk called 'ramcart'
1080 :
1090 open 15,8,15: open 8,8,1, "0:ramcart"
1100 input#15,e,e$,b,c: if e then close 15: print e;e$;b;c:
     stop
1110 for j = 32768 to 32999: read x: print#8,chr$(x);:
     ch = ch + x: next: close8
1120 if ch<>28345 then print "checksum error!": stop
1130 print "** module created **": end
1140 :
1150 data   0, 128,   9, 128,  94, 254, 195, 194
1160 data 205,  56,  48, 162,   5, 142,  22, 208
1170 data  32, 163, 253,  32,  80, 253,  32,  21
1180 data 253,  32,  91, 255,  88,  32,  83, 228
1190 data  32, 191, 227, 162, 251, 154, 172, 224
1200 data 128, 174, 225, 128, 132,  43, 134,  44
1210 data 172, 228, 128, 174, 229, 128, 132,  95
1220 data 134,  96, 172, 226, 128, 174, 227, 128
1230 data 132,  88, 134,  89, 136, 192, 255, 208
1240 data   1, 202, 132,  45, 134,  46, 169, 160
1250 data 133,  91, 169,   0, 133,  90,  32, 191
1260 data 163, 169,  82, 141, 119,   2, 169,  85
1270 data 141, 120,   2, 169,  78, 141, 121,   2
1280 data 169,  13, 141, 122,   2, 169,   4, 133
1290 data 198, 108,   2,   3,  56, 165,  46, 229
1300 data  44, 170, 165,  45, 229,  43, 168, 224
1310 data  31, 176,  67, 140, 228, 128, 142, 229
1320 data 128,  56, 169, 159, 237, 229, 128, 141
1330 data 229, 128, 169, 255, 237, 228, 128, 141
1340 data 228, 128, 165,  43, 141, 224, 128, 133
1350 data  95, 165,  44, 141, 225, 128, 133,  96
1360 data 164,  45, 166,  46, 200, 208,   1, 232
1370 data 140, 226, 128, 132,  90, 142, 227, 128
1380 data 134,  91, 169, 160, 133,  89, 169,   0
1390 data 133,  88,  32, 191, 163,  96, 169, 204
1400 data 160, 128,  32,  30, 171,  96,  80,  82
1410 data  79,  71,  82,  65,  77,  32,  84,  79
1420 data  79,  32,  76,  65,  82,  71,  69,  10
1430 data  13,   0,   0,   0,   0,   0,   0,   0
```

Listato 2

```
1000 rem save "0:ramcart.pal",8
1010 rem ** by: john bush and noel nyman - seattle, wa
1020 rem ** auto-start support prg for c64 ram cartridge
1030 :
1040 open 8,8,1, "0:ramcart"
1050 sys 700
1060 .opt o8
1070 *          =      $8000
1080 ;
1090 ;*** equates ***
1100 ;
1110 txttab   =   $2b     ;start of basic text
1120 vartab   =   $2d     ;end of basic text
1130 source   =   $5f     ;start of source to copy
1140 end      =   $5a     ;end + 1 of source to copy
1150 dest     =   $58     ;end + 1 of destination
1160 ndx      =   $c6     ;no of characters in keyboard
                           buffer
1170 keyd     =   $0277   ;start of keyboard buffer
1180 warm     =   $0302   ;basic warm start vector
1190 copy     =   $a3bf   ;copy memory
1200 strout   =   $ab1e   ;print string
1210 vicctrl  =   $d016   ;vic control register
1220 vectors  =   $e453   ;copy basic vectors to ram
1230 init     =   $e3bf   ; initialize basic interpreter
1240 ioinit   =   $fda3   ;initialize i/o
1250 ramtas   =   $fd50   ;initialize memory pointers
1260 restor   =   $fd15   ;restore i/o vectors
1270 cint     =   $ff5b   ;init screen and keyboard
1280 nmicont  =   $fe5e   ;continue with nmi routine
1290 ;
1300 ;*** auto-start basic program ***
1310 ;
1320 ;place start of code in cartridge vectors
1330 .byte    <start,>start
1340 .byte    <nmicont,>nmicont
1350 ; 'cbm' with bit 7 set
1360 .byte    $c3,$c2,$cd
1370 .asc     "80"
1380 ;
1390 ;'start' calls most of the routines
1400 ;which would be executed if a cartridge
1410 ;had not been detected. system vectors
1420 ;and basic are initialized.
1430 ;
1440 start  ldx   #5
1450        stx   vicctrl
1460        jsr   ioinit
1470        jsr   ramtas
1480        jsr   restor
1490        jsr   cint
1500        cli
1510        jsr   vectors
1520        jsr   init
1530        ldx   #$fb
1540        txs              ;initialize stack pointer
1550 ;
1560 ;copy the basic program from
1570 ;the area under $a000 to the start-of-basic
1580 ;and set up the basic text and variables
1590 ;vectors. place 'run' in the keyboard buffer and
1600 ;enter basic through the warm start vector.
1610 ;
1620        ldy   txtt       ;store start of basic
1630        ldx   txtt + 1   ;saved with program
```

```
1640          sty   txttab      ;at op system vector
1650          stx   txttab + 1
1660          ldy   stsour      ;store start of source
1670          ldx   stsour + 1  ;at vector for copy routine
1680          sty   source
1690          stx   source + 1
1700          ldy   vart        ;store end of destination
                                (+ 1)
1710          ldx   vart + 1    ;at copy routine vector
1720          sty   dest
1730          stx   dest + 1
1740          dey               ;subtract one from low byte
1750          cpy   #$ff
1760          bne   cont
1770          dex               ;subtract borrow
1780 cont     sty   vartab      ;store op system vector
1790          stx   vartab + 1
1800          lda   #$a0        ;end of source (+ 1) =
                                $a000
1810          sta   end + 1
1820          lda   #0
1830          sta   end
1840          jsr   copy
1850          lda   # " r "
1860          sta   keyd
1870          lda   # " u "
1880          sta   keyd + 1
1890          lda   # " n "
1900          sta   keyd + 2
1910          lda   #$0d        ;<return>
1920          sta   keyd + 3
1930          lda   #4          ;number of characters
1940          sta   ndx
1950          jmp   (warm)
1960 ;
1970 ;*** store basic program to cartridge ***
1980 ;calculate the size of the basic text, and
1990 ;print an error message if too large to fit
2000 ;in the cartridge. if okay, subtract the size
2010 ;from $9fff to get the location of the start
2020 ;of the copy to be saved to cartridge. save
2030 ;that vector, and the start and end of basic
2040 ;text for future use. set-up vectors for
2050 ;copy routine and copy program to cartridge.
2060 ;
2070 store    sec
2080          lda   vartab + 1
2090          sbc   txttab + 1  ;find size of basic program
2100          tax
2110          lda   vartab
2120          sbc   txttab
2130          tay
2140          cpx   #$1f        ;max size allowed
2150          bcs   error       ;print error message and
                                quit
2160          sty   stsour      ;store size temporarily
2170          stx   stsour + 1
```

```
2180          sec
2190          lda   #$9f        ;subtract size from $9fff to
                                find
2200          sbc   stsour + 1  ;start of program in car-
                                tridge memory
2210          sta   stsour + 1
2220          lda   #$ff
2230          sbc   stsour
2240          sta   stsour
2250          lda   txttab      ;store start of basic for
                                cartridge
2260          sta   txtt        ;use and in vector for copy
                                routine
2270          sta   source
2280          lda   txttab + 1
2290          sta   txtt + 1
2300          sta   source + 1
2310          ldy   vartab      ;store end of basic (+ 1) for
                                cartridge
2320          ldx   vartab + 1  ;use and vector for copy
                                routine
2330          iny
2340          bne   cont1
2350          inx
2360 cont1    sty   vart
2370          sty   end
2380          stx   vart + 1
2390          stx   end + 1
2400          lda   #$a0        ;store $a000 (end of car-
                                tridge memory + 1)
2410          sta   dest + 1    ;in vector for read routine
2420          lda   #0
2430          sta   dest
2440          jsr   copy
2450          rts
2460 ;
2470 ;*** print error message ***
2480 ;
2490 error    lda   #<message
2500          ldy   #>message
2510          jsr   strout
2520          rts
2530 ;
2540 message  *
2550 .asc     " program too large "
2560 .byte    $0a,$0d,$00
2570 ;
2580 ;*** system vector storage ***
2590 ;
2600 txtt     .word 0           ;start of program in ram
2610 vart     .word 0           ;end of program in ram
2620 stsour   .word 0           ;start of source in cartridge
2630 ;
2640 .end
```